

Basic XML Schema for NIEM



Modules Roadmap:

You Are Here

Anatomy of an XML Exchange

XML Conceptual Review



Basic XML Schema for NIEM

Advanced XML Schema for NIEM

Substitution Groups

Extension Schemas

Objectives Roadmap

This module supports the following course objectives:



Define the physical components of an XML exchange.



Identify basic XML components that are used in the NIEM structure.



Write and/or extend an XML schema conformant to the NIEM Naming and Design Rules (NDR).

Module Objectives

- After completing this module, you should be able to:
 - ◆ Compare and contrast locally defined types, elements and attributes against globally defined ones.
 - ◆ Compare and contrast anonymous types against named types.
 - ◆ Demonstrate the reuse of named types.
 - ◆ Explain the purpose of annotations.
 - ◆ Differentiate between complex types with simple content verses complex type with complex content.

XML Schema Allows Great Flexibility

- Not all constructs allowed within XML schema are allowed in NIEM.
 - ◆ XML schema is extremely flexible.
 - ◆ NIEM needs constraints to lock down semantic meanings of data.
 - ◆ All disallowed constructs with explanations are contained in the NIEM Naming and Design Rules (NDR).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="Person" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PersonName">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PersonGivenName" type="xs:string"/>
              <xs:element name="PersonMiddleName" type="xs:string" minOccurs="..."/>
              <xs:element name="PersonSurName" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
<xs:attribute name="personNameCode" use="optional" default="RealName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="RealName"/>
      <xs:enumeration value="Alias"/>
      <xs:enumeration value="BirthName"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element >
</xs:schema>
```

Specifies root element

Child Element #1 of Person
- Complex Content Model

Child Elements of PersonName -
Simple Content without Facets

Truncated because
of slide real estate

Attribute of Person -
Simple Content Model
implementing Facets

Instance Review

- Example XML Instance of previous slide

```
<?xml version="1.0" encoding="UTF-8"?>
<Person personNameCode="BirthName">
  <PersonName>
    <PersonGivenName>Bart</PersonGivenName>
    <PersonMiddleName>Jojo</PersonMiddleName>
    <PersonSurName>Simpson</PersonSurName>
  </PersonName>
</Person>
```

Schema For Use With Exercise 3.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PersonName">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PersonGivenName" type="xs:string"/>
              <xs:element name="PersonSurName" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="personNameCommentText" use="optional" type="xs:string"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="PersonUSCitizenIndicator" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exercise 3.1: Schema

- Using the schema from the previous slide:
 - ◆ How many root elements are there and what are they?
 - ◆ List the elements of complex types in the schema.
 - ◆ List the elements simple types in the schema.
 - ◆ List the attributes in the schema.

Solution 3.1: Schema

- How many root elements are there?
 - ◆ Just one: Person
- List the elements of complex types in the schema
 - ◆ Person
 - ◆ PersonName
- List the elements of simple types in the schema
 - ◆ PersonGivenName (string)
 - ◆ PersonSurName (string)
 - ◆ PersonNameCommentText (string)
 - ◆ personUSCitizenshipIndicator (boolean)
- List the attributes in the schema
 - ◆ personNameCommentText

Element Cardinality

- An XML schema defines the number of child elements (quantity) permitted or required in a document.
- minOccurs
- maxOccurs
- default = 1 if not declared
- Can be declared as follows:
 - ◆ Define minOccurs: 0, 1
 - ◆ Define maxOccurs: 0, 1, or "unbounded"

Declarations: Global vs. Local

- Globally defined elements, attributes, and types are globally reusable.
 - ◆ Ensures that name duplication does not occur within a namespace.
 - ◆ Allows for maximum reuse.
- Locally defined elements, attributes, and types are *not* reusable outside the context in which they are defined.

**NIEM uses and allows only
global declarations**

Local Element Declaration

```
<xs:complexType name="HospitalType">  
  <xs:sequence>  
    <xs:element name="HospitalName" type="xs:string" />  
  </xs:sequence>  
</xs:complexType>
```

Locally defined
elements

Not used
in NIEM!

```
<xs:complexType name="SchoolType">  
  <xs:sequence>  
    <xs:element name="SchoolName" type="xs:string" />  
  </xs:sequence>  
</xs:complexType>
```

Global Element Declaration

```
<xs:complexType name="HospitalType">  
  <xs:sequence>  
    <xs:element ref="Name"/>  
  </xs:sequence>  
</xs:complexType>
```

```
<xs:complexType name="SchoolType">  
  <xs:sequence>  
    <xs:element ref="Name"/>  
  </xs:sequence>  
</xs:complexType>
```

```
<xs:element name="Name" type="xs:string"/>
```

References to the
globally defined
element

Globally Defined
Element

Cardinality for
globally defined
elements is always
min and max of 1

Anonymous Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PersonName">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PersonGivenName" type="xs:string"/>
              <xs:element name="PersonSurName" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="personNameCommentText" use="optional" type="xs:string"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="PersonUSCitizenIndicator" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Types defined hierarchically without explicit naming

Not used in NIEM!

Named Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
  <xsc:complexType name="PersonType">
    <xsc:sequence>
      <xsc:element ref="PersonName"/>
      <xsc:element ref="PersonUSCitizenIndicator"/>
    </xsc:sequence>
  </xsc:complexType>
  <xsc:complexType name="PersonNameType">
    <xsc:sequence>
      <xsc:element ref="PersonGivenName"/>
      <xsc:element ref="PersonSurName"/>
    </xsc:sequence>
    <xsc:attribute ref="personNameCommentText" use="optional"/>
  </xsc:complexType>
  <xsc:element name="Person" type="PersonType"/>
  <xsc:element name="PersonName" type="PersonNameType"/>
  <xsc:element name="PersonGivenName" type="xs:string"/>
  <xsc:element name="PersonSurName" type="xs:string"/>
  <xsc:attribute name="personNameCommentText" type="xs:string"/>
  <xsc:element name="PersonUSCitizenIndicator" type="xs:boolean"/>
</xss:schema>
```

Complex Types defined globally and explicitly named

Local elements reference the globally declared ones

Elements are globally declared. All can be re-used.

Exercise 3.2: Create Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<Incident incidentCategory="Miscellaneous">
  <IncidentDescription>
    Something happened
  </IncidentDescription>
  <IncidentPrimaryLocation>
    <Address>
      <StreetAddress>1313 Mockingbird Lane</StreetAddress>
      <City>Mockingbird Heights</City>
    </Address>
  </IncidentPrimaryLocation>
  <IncidentSecondaryLocation>
    <Address>
      <StreetAddress>742 Evergreen Terrace</StreetAddress>
      <City>Springfield</City>
    </Address>
  </IncidentSecondaryLocation>
</Incident>
```

Solution 3.2: Schema *(1 of 2)*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsc:complexType name="AddressType">
    <xss:sequence>
      <xselement ref="StreetAddress"/>
      <xselement ref="City"/>
    </xss:sequence>
  </xsc:complexType>
  <xsc:complexType name="IncidentType">
    <xss:sequence>
      <xselement ref="IncidentDescription"/>
      <xselement ref="IncidentPrimaryLocation"/>
      <xselement ref="IncidentSecondaryLocation"/>
    </xss:sequence>
    <xsa:attribute ref="incidentCategory" use="required"/>
  </xsc:complexType>
```

Solution 3.2: Schema (2 of 2)

```
<xs:complexType name="LocationType">
  <xs:sequence>
    <xs:element ref="Address"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Address" type="AddressType"/>
<xs:element name="City" type="xs:string"/>
<!-- Incident is the root element -->
<xs:element name="Incident" type="IncidentType"/>
<xs:attribute name="incidentCategory" type="xs:string"/>
<xs:element name="IncidentDescription" type="xs:string"/>
<xs:element name="IncidentPrimaryLocation" type="LocationType"/>
<xs:element name="IncidentSecondaryLocation" type="LocationType"/>
<xs:element name="StreetAddress" type="xs:string"/>
</xs:schema>
```

Schema Annotations

- Used extensively in NIEM to convey:
 - ◆ Semantic meaning for data elements.
 - ◆ Type application structure information to the schema.
- Two specific schema annotation tags:
 - ◆ documentation
 - Add descriptive information about the element, attribute or type.
 - ◆ appinfo
 - Add specific instructions for schema based tools.

Annotations

```
<xs:complexType name="PersonNameType">
  <xs:annotation>
    <xs:documentation>
      A data type that defines the structure of a person's name
    </xs:documentation>
    <xs:appinfo>External application info goes here</xs:appinfo>
  </xs:annotation>
  <xs:sequence>
    <xs:element ref="PersonGivenName"/>
    <xs:element ref="PersonMiddleName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="PersonSurName"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="PersonName" type="PersonNameType">
  <xs:annotation>
    <xs:documentation>The birth name of a person</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="PersonAliasName" type="PersonNameType"/>
```

Defines the precise semantic meaning of the tag



Content Models

- Every complex type in XML schema has a content model.
- Content models can be simple or complex.
- Simple content models either add attributes to or restrict from a simple type.
- Complex content models specify child elements.

Simple Content Model

- One kind of simple content model adds attributes to a simple type

```
<xs:complexType name="VehicleSpeed">  
  <xs:simpleContent>  
    <xs:extension base="xs:int">  
      <xs:attribute name="units" type="xs:string"/>
```

- Another kind of simple content model restricts a simple type

```
<xs:complexType name="StringOfFortyChars">  
  <xs:simpleContent>  
    <xs:restriction base="xs:string">  
      <xs:maxLength value="40"/>
```

Complex Content Model

- Complex content models specify “child” structures beneath an element.
- Three choices (called “compositors”):
 - ◆ xs:sequence (ordered list of elements)
 - ◆ xs:choice (choice of one element among many)
 - NIEM Naming and Design Rules is currently ambiguous about restricting use.
 - ◆ xs:all (unordered list of elements)
 - Completely disallowed in NIEM.

Complex Content Model Examples

Schema Fragments

```
<xs:element name="e" type="CT"/>
<xs:complexType name="CT">
  <xs:sequence>
    <xs:element ref="child1"/>
    <xs:element ref="child2"/>
```

Instance Fragments

```
<e>
  <child1/>
  <child2/>
</e>
```

```
<xs:element name="e" type="CT"/>
<xs:complexType name="CT">
  <xs:choice>
    <xs:element ref="child1"/>
    <xs:element ref="child2"/>
```

```
<e>
  <child1/>
</e>
```

or

```
<e>
  <child2/>
</e>
```

- Avoid xs:all

Module Summary

- After completing this module, you should be able to:
 - ◆ Compare and contrast locally defined types, elements and attributes against globally defined ones.
 - ◆ Compare and contrast anonymous types against named types.
 - ◆ Demonstrate the reuse of named types.
 - ◆ Explain the purpose of annotations.
 - ◆ Differentiate between complex types with simple content verses complex type with complex content.

Creative Commons



Attribution-ShareAlike 2.0

You are free to

- Copy, distribute, display, and perform the work
- Make derivative works
- Make commercial use of the work



Attribution—You must give the original author credit



ShareAlike—If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one

Under the following conditions

- For any reuse or distribution, you must make clear to others the license terms of this work
- Any of these conditions can be waived, if you get permission from the copyright holder

Your fair use and other rights are in no way affected by the above

This is a human-readable summary of the [Legal Code \(the full license\)](#) and [Disclaimer](#)

This page is available in the following languages

[Català](#), [Deutsch](#), [English](#), [Castellano](#), [Suomeksi](#), [français](#), [hrvatski](#), [Italiano](#), [日本語](#), [Nederlands](#), [Português](#), and [中文\(繁\)](#)

[Learn how to distribute your work using this license](#)